VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

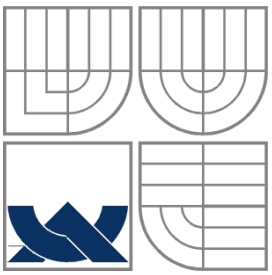
USB KLÍČ

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

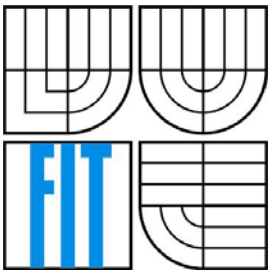
AUTOR PRÁCE
AUTHOR

FRANTIŠEK SLIMAŘÍK

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

USB KLÍČ
USB KEY

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE
AUTHOR

FRANTIŠEK SLIMAŘÍK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JOSEF STRNADEL, Ph.D.

BRNO 2008

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2007/2008

Zadání bakalářské práce

Řešitel: **Slimařík František**
Obor: Informační technologie
Téma: **USB klíč**
Kategorie: Vestavěné systémy

Pokyny:

1. Seznamte s mechanismy zabezpečování pomocí tzv. USB klíčů
2. Po dohodě s vedoucím navrhněte architekturu USB klíče
3. Realizujte prototyp USB klíče navrženého v předchozím bodě
4. Funkci USB klíče demonstrujte na jednoduchých aplikacích

Literatura:

- *EZK - elektronika Zdeněk Krčmář* [online]. c2007. http://www.ezk.cz/.
- *FLAJZAR... výroba a prodej elektroniky* [online]. c2005. http://www.flajzar.cz/.
- *Freescale Semiconductor* [online]. c2007. http://www.freescale.cz/.
- *GME Česko* [online]. c2007. http://www.gme.cz/.
- *USB.cz - informace o USB zařízení.* [online]. c2007. www.usb.cz.

Při obhajobě semestrální části projektu je požadováno:

1. Seznamte s mechanismy zabezpečování pomocí tzv. USB klíčů
2. Po dohodě s vedoucím navrhněte architekturu USB klíče

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Strnadel Josef, Ing., Ph.D.**, UPSY FIT VUT
Datum zadání: 1. listopadu 2007
Datum odevzdání: 14. května 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

L.S.

doc. Ing. Zdeněk Kotásek, CSc.
vedoucí ústavu

Licenční smlouva

Licenční smlouva je uložena v archivu Fakulty informačních technologií Vysokého učení technického v Brně.

Abstrakt

USB rozhraní patří mezi dnes nejpoužívanější rozhraní pro komunikaci mezi počítačem a externími zařízeními. Musí splňovat striktní požadavky, aby byla zajištěna jeho multiplatformnost. Postupem času nalezlo toto rozhraní své uplatnění i v mikrokontrolérech. Práce se zabývá návrhem, sestavením a naprogramováním hardwarového klíče USB standardu třídy HID s využitím mikrokontroléru MC68HC908JB8.

Klíčová slova

USB rozhraní, HC08, USB klíč, hardware, zařízení standardu HID

Abstract

These days USB interface belongs to the most used interfaces for communication between computer and external devices. It must satisfy strict requirements to be provided its multiplatformity. In the process this interface found its use even in microcontrollers. This thesis is focused on design, compilation and programming of hardware USB key of HID standard with usage of microcontroller MC68HC908JB8.

Keywords

USB interface, HC08, USB key, hardware, HID standard device

Citace

František Slimařík: USB klíč, bakalářská práce, Brno, FIT VUT v Brně, 2008

USB klíč

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Josefa Strnadela, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
František Slimařík
1. května 2008

Poděkování

Rád bych poděkoval vedoucímu mé bakalářské práce Ing. Josefu Strnadelovi, Ph.D., za podnětné připomínky a odborné rady.

Obsah

Úvod

Spolu s rostoucím počtem počítačů využívaných jak v domácnostech tak ve firmách a organizacích rostl i počet periferních zařízení, která mohou s počítačem spolupracovat. Každé takovéto zařízení bylo nutné připojit k tomuto účelu specifické sběrnici a nakonfigurovat jej. Samotný počítač se před touto činností musel většinou vypnout. Nastala tedy potřeba zavedení pro uživatele jednoduchého a rychlého rozhraní pro připojení více periferií a to za chodu počítače. Těmto požadavkům plně vyhovuje standard USB. S masivním rozšířením zařízení využívajících toto rozhraní se dostalo i na podporu USB přenosů ze stran mikrokontrolérů.

Stejně jako růst počítačové technologie ovlivnil jednotlivá rozhraní a pohled na ně, bylo nutné zajistit bezpečnost uživatelských aplikací a dat. Počítače se využívají téměř ve všech odvětvích a jako takové přichází do kontaktu s citlivými informacemi. Začínají se objevovat postupy pro šifrování dat, používají se hesla, filtrují se přístupy k datům a našli bychom mnoho dalších způsobů, jak předejít nežádoucímu čtení soukromých informací.

V souvislosti s výše zmíněným jsem se rozhodl realizovat jednoduchý hardwarový USB klíč pro demonstraci zabezpečování aplikací. Využil jsem tedy rozhraní USB popsaného v kapitole 1 a jako cílovou platformu zvolil rodinu mikrokontrolérů Motorola s jádrem HC08, jimž se věnuje kapitola 2. Kapitola 3 popisuje vlastnosti hardwarového zařízení, možnosti komunikace s cílovým operačním systémem i samotný návrh klíče. Kapitola 4 se zabývá podrobným popisem firmwaru, který řídí činnost navrženého zařízení. V kapitole 5 jsou popsány testovací úlohy a jejich implementace prostřednictvím volání API funkcí operačního systému Microsoft Windows.

1 Popis rozhraní USB

USB rozhraní (Universal Serial Bus) se během posledních let stalo zcela běžnou součástí spotřební elektroniky připojitelné k počítači a již téměř vytlačilo klasický sériový port RS232 a někde dokonce i port paralelní. Jedná se o typ rozhraní používaného pro periferní zařízení jako jsou tiskárny, klávesnice a mnoho dalších. Detailní specifikace je popsána na www stránkách [7].

1.1 Základní vlastnosti

Mezi základní vlastnosti standardu USB patří především rychlost přenosu informací. USB verze 1.1 podporuje dvě rychlosti, full-speed (12 Mbit/s) a low-speed (1,5 Mbit/s). Nová verze tohoto standardu, USB 2.0, podporuje rychlost až 480 Mbit/s. V současné době však ani tato rychlost není dostačující a tudíž je očekáván další nástupce, USB 3.0, který by měl přinést rychlost dosahující až 4,8 Gbit/s.

Zařízení na sběrnici USB jsou řízena hostitelem. Podle původní specifikace se v celém systému smí vyskytovat pouze jeden takovýto hostitel. S novým standardem se však objevil i tzv. HNP protokol (Host Negotiation Protocol), který podporuje hostitele dva. V takovémto případě se jedná o dvě nezávislé USB sběrnice. Hostitel je zodpovědný za přebírání všech transakcí a plánování šířky pásma. Data mohou být zasílána různými transakčními metodami využívajícími protokol založený na tokenech. USB hostitelé mají svou vlastní specifikaci. U USB 1.1 jsou to standardy UHCI (Universal Host Controller Interface), vytvořený společností Intel, a OHCI (Open Host Controller Interface), vytvořený spoluprácí firem Compaq, Microsoft a National Semiconductor. U standardu USB 2.0 pak spoluprácí výše zmiňovaných firem vznikla specifikace EHCI (Enhanced Host Controller Interface).

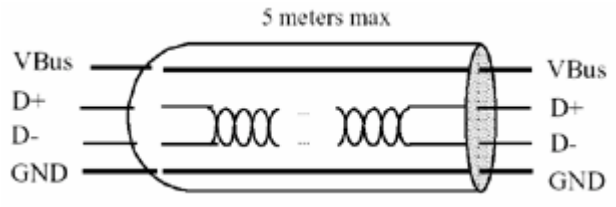
USB používá topologii hvězdy podobnou topologii, kterou využívá 10BaseT Ethernet. Na jedinou sběrnici můžeme připojit až 127 různých zařízení. V případě potřeby můžeme využít rozbočovačů (hub) a počet připojených zařízení zvýšit. Rozbočovač vytváří rozšiřující porty a chová se jako běžné zařízení, které obsadí jeden port. Zároveň však vytvoří několik dalších plnohodnotných portů.

Další z výhod této technologie je podpora Plug-and-Play. Ta umožňuje uživateli připojit zařízení za chodu PC. Hostitel detekuje zařízení, integruje jej do systému a načte pro něj příslušný ovladač. Při odpojení hostitel detekuje jeho absenci a automaticky ovladač odstraní z paměti. K takovéto detekci je využito kombinace PID/VID (Product ID a Vendor ID).

1.2 Elektrické vlastnosti

USB rozhraní využívá dva typy konektorů. Plochý konektor typ A je dnes obsažen na prakticky každém novém PC v minimálně 2 konektorech (některé základní desky mají integrován rovnou USB hub, který obsahuje až 8 portů přímo v PC). Druhý konektor, typ B, je určen pro periferní zařízení, čímž je zároveň definován standard propojovacího kabelu. Typy konektorů jsou uvedeny na obrázku č. 1.1.

Obrázek 1.1: USB hostitel (A), USB zařízení (B)

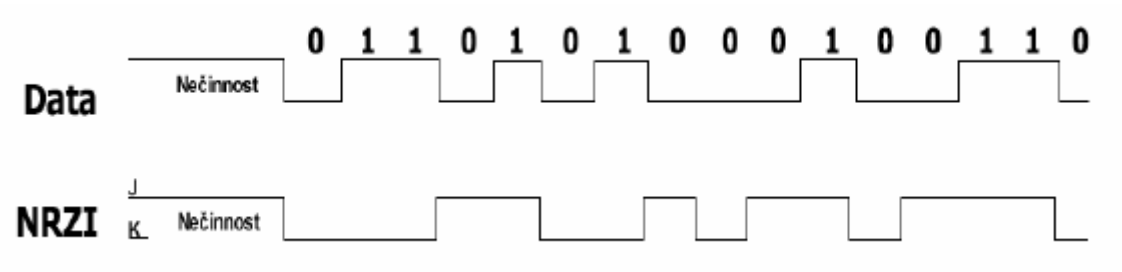


Obrázek 1.2: USB kabel

Pro přenos dat a napájecího napětí (5V) se užívá kabel se čtyřmi vodiči znázorněný na obrázku č. 1.2. Jak je z obrázku patrné, data jsou přenášena po dvou kroucených vodičích. Spolehlivý přenos informací je založen na principu NRZI. Data jsou vysílána sériově a vždy při načtení logické nuly dojde k překlopení logické hodnoty. Pokud však nedojde po delší dobu ke změně signálu, může nastat desynchronizace hodin vysílače a přijímače. Typickým příkladem této situace je přenos velkého množství logických „1“, které výstupní signál nemodifikují. Abychom tomu předcházeli, je využito metody vkládání (bit stuffing), která po odeslání šesti „1“ automaticky vloží logickou nulu. Všechny redundantní informace jsou poté na straně přijímače odstraněny.

Proud, který můžeme ze sběrnice odebírat, je omezen. Informace o velikostech napětí přenášených logických úrovní, proudové zátěži a přípustných tolerancích je možné nalézt na www stránkách [7] nebo [4]. Podle způsobu napájení rozlišujeme dvě skupiny zařízení:

- Zařízení napájená pouze ze sběrnice (Bus-Powered).
- Zařízení s externím zdrojem (Self-Powered).



Obrázek 1.3: Princip přenosu dat

1.2.1 Režim s nízkou spotřebou

USB zařízení umožňují přechod do režimu s nízkou spotřebou (suspend mode). Do tohoto režimu se zařízení dostane v případě, že datová linka je v nečinném stavu po dobu delší než 3 ms. Jakákoliv aktivita na datové lince přivede zařízení zpět do normálního režimu. Podle USB specifikace zařízení napájená ze sběrnice vyžadují méně než 500 μA odebíraného proudu ze zdroje, pokud se nachází v režimu s nízkou spotřebou.

1.3 Principy komunikace

Na rozdíl od komunikace prostřednictvím sériové linky je USB standard tvořen několika vrstvami protokolů. Každá transakce prostřednictvím USB komunikace je tvořena:

- Řídícím paketem (token packet).
- Datovým paketem.
- Stavovým paketem (chybová hlášení, potvrzení).

Jak bylo uvedeno výše, USB standard využívá hostitelem ovládanou sběrnici. Hostitel inicializuje všechny transakce zařízení. První paket, zvaný token, popisuje jakým způsobem se bude komunikace ubírat, bude-li se číst nebo zapisovat, s kterým zařízením se bude pracovat a jaký koncový bod bude využit. Následující paket je většinou datový. Po tomto paketu je zaslán paket stavový obsahující informace o doručení předešlého paketu (data došla v pořádku, data nebyla k dispozici).

1.3.1 Popis USB paketu

USB sběrnice přenáší jako první nejméně významný bit (LSB). USB pakety jsou tvořeny následujícími poli:

- **Sync** – Všechny pakety musí začínat tímto polem. Velikost tohoto pole je 8 bitů, které jsou využity pro synchronizaci vysílače a přijímače. Poslední dva bity indikují, kde začíná PID.
- **PID -** Pole PID (Packet ID) je použito pro identifikaci typu paketu, jež byl zaslán. Jednotlivé typy paketů jsou uvedeny na obrázku 1.4. První 4 bity jednoznačně identifikují paket, zatímco následující 4 jsou komplementy těchto bitů.
- **ADDR** – Pole adres specifikuje, pro které zařízení je paket určen. Toto pole bývá sedmibitové, čímž je umožněno připojit až 127 zařízení takto adresovatelných. Po připojení zařízení do systému je automaticky jeho adresa nastavena na 0. Díky tomu hostitel rozpozná nové zařízení.
- **ENDP** – Pole se čtyřmi bity umožňující popsat 16 různých typů koncových bodů (endpoints).
- **CRC** – Cyklická redundantní kontrola. Všechny řídící pakety mají 5bitovou CRC, zatímco pakety datové 16bitovou.
- **EOP** – Signalizuje konec paketu.

Group	PID Value	Packet Identifier
Token	0001	OUT Token
	1001	IN Token
	0101	SOF Token
	1101	SETUP Token
Data	0011	DATA0
	1011	DATA1
	0111	DATA2
	1111	MDATA
Handshake	0010	ACK Handshake
	1010	NAK Handshake
	1110	STALL Handshake
	0110	NYET (No Response Yet)
Special	1100	PREamble
	1100	ERR
	1000	Split
	0100	Ping

Obrázek 1.4: Typy USB paketů

1.3.2 Typy USB paketů

USB podporuje čtyři různé typy paketů. Token paket indikující typ transakce, datový paket s informacemi, paket užitý pro potvrzení a hlášení chyb (handshake) a paket pro začátek rámce (využito u isochronních přenosů). Pakety popsané níže jsou využívaný v USB standardu 1.1, tudíž jejich výčet není úplný. Formát vyjmenovaných paketů je popsán na obrázku č. 1.5.

- **Token paket**

  Existují celkem tři typy těchto paketů:

  - **IN** – Informuje USB zařízení o žádosti hostitele číst informace.
  - **OUT** – Informuje USB zařízení o žádosti hostitele zapisovat informace.
  - **SETUP** – Použito pro zahájení přenosu.
- **Datový paket**

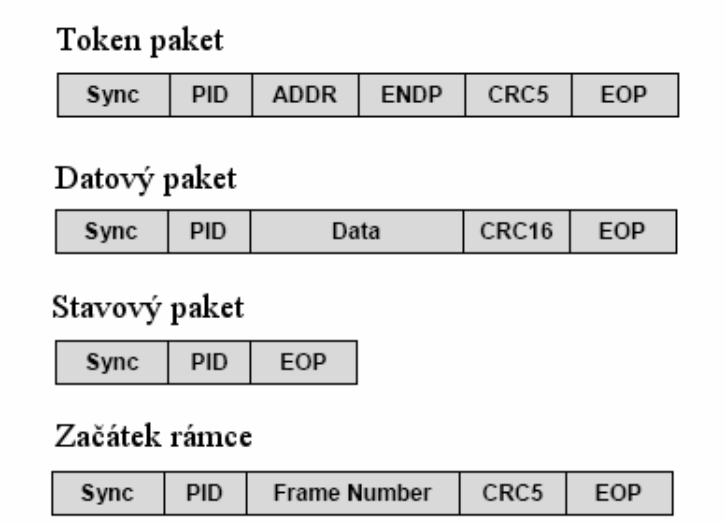
  Existují dva typy datových paketů, z nichž každý je schopen přenést až 1023 bytů dat:

  - **DATA0**
  - **DATA1**
- **Stavový paket**

  Existují tři typy stavových paketů:

  - **ACK** – Potvrzení o úspěšném doručení paketu.
  - **NAK** – Zpráva o tom, že zařízení nemůže dočasně přijmout či odeslat data.
  - **STALL** – Zařízení shledalo, že je ve stavu, který vyžaduje zásah od hostitele.
- **Paket určující začátek rámce**

  Tento paket je tvořen jedenáctibitovým číslem rámce (frame number) zasílaným hostitelem každou 1 milisekundu.



Obrázek 1.5: Formát USB paketů

1.3.3 Koncové body

Koncové body můžeme popsat jako zdroje nebo úložiště dat, které tvoří konec komunikačního kanálu. Hostitel odešle data a ty skončí v jednom z těchto koncových bodů. Pokud chce data odeslat zařízení, nemůže jednoduše data poslat do počítače, jelikož je sběrnice řízená hostitelem. Na místo

toho data zapíše do koncového bodu (např. EP1 IN), ve kterém data setrvají, dokud si je hostitel sám nevyžádá.

Koncové body můžeme vnímat jako rozhraní mezi hardwarovým zařízením a firmwarem, který na tomto zařízení běží. Všechna zařízení musí podporovat nultý koncový bod EP0. Tento koncový bod přijímá všechny kontrolní a stavové žádosti při operacích na sběrnici.

1.3.4 Roury

Zatímco zařízení vysílá a přijímá data sérií koncových bodů, klientský software využívá k přenosům tzv. roury (pipes). Roura tvoří logické spojení mezi hostitelem a koncovým bodem. Roura jako taková má množství parametrů, které musí být nastaveny pro správný průběh komunikace. Typickým příkladem je obousměrná roura tvořící spojení s nultým koncovým bodem pro přenos dat z a do zařízení. USB standard definuje dva typy rour:

- **Proudové roury** (stream pipes), které nemají definovaný formát a umožňují poslat jakýkoliv typ dat. Data přicházejí sekvenčně a mají předdefinovaný směr. Tento druh rour podporuje nárazové (bulk), izochronní a přerušované typy toků dat. Mohou být řízeny hostitelem nebo zařízením.
- **Roury zpráv** (message pipes) mají definovaný formát. Jsou řízeny hostitelem na základě žádosti přijaté od zařízení. Data mohou být přenášena v obou směrech, ale jsou podporovány pouze řídící toky dat.

1.3.5 Typy koncových bodů

Hostitel je zodpovědný za řízení přenosu na sběrnici. To se děje při počáteční inicializaci, při konfiguraci isochronních a přerušovacích koncových bodů a během operací na sběrnici. U full-speed zařízení je periodický přenos (isochronní a přerušovací) limitován 90% šířkou pásma, zatímco u zařízení high-speed je to již 80 %. Z těchto faktů vyplývá, že zbylá část je vyhrazena pro řídící přenosy a minimální podíl z této části pak pro přenosy nárazové. Popis jednotlivých přenosů je uveden níže.

1.3.5.1 Řídící toky dat (Control Transfers)

Typicky jsou používány pro příkazové a stavové operace a vytvářejí základ pro nastavení USB zařízení. Délka paketu řídícího datového toku musí být 8 bytů u low-speed zařízení, zatímco u zařízení high-speed jsou povoleny délky 8, 16, 32 a 64 bytů. Tento přenos je rozdělen do tří fází.

- Fáze nastavení (**Setup Stage**) se skládá ze tří paketů. Nejprve je zaslán SETUP token paket obsahující adresu zařízení a adresu koncového bodu. Ten je následován datovým paketem, který

vždy obsahuje PID DATA0 a detaily o nastavení přenosu. Nakonec je zaslán stavový paket pro potvrzení úspěšného průběhu nebo s informacemi o chybách.

- Fáze datová (**Data Stage**) je tvořena jedním nebo několika IN nebo OUT přenosy. Nastavení vyžaduje definování množství dat přenášených v této fázi. Pokud velikost dat, která potřebujeme přenést, přesáhne maximální velikost paketu, bude použit vícenásobný přenos, ve kterém každý paket bude maximálně využit až do posledního přenášeného. Datová fáze má dvě možné varianty závislé na směru přenosu.
  - **IN:** Jestliže je hostitel připraven přijmout data, využije se token IN. Pokud je tento token přijat chybně, je ignorován. V případě, že je přijat správně, zařízení může odpovědět datovým paketem obsahujícím řídící data, paketem STALL indikujícím chybu nebo paketem NAK, oznamujícím, že koncový bod právě nemá data k dispozici.
  - **OUT:** Pokud hostitel potřebuje odeslat data do zařízení, využije tokenu OUT následovaného datovým paketem. Pokud při přenosu vznikne chyba, je opět token i datový paket ignorován. Jestliže byl koncový bod prázdný a data byla v pořádku doručena, je zasláno potvrzení o úspěšném doručení (ACK). Pokud by koncový bod prázdný nebyl, bylo by to oznámeno prostřednictvím paketu NAK. V případě chyby pak opět přichází na řadu paket STALL.
- Fáze stavová (**Status Stage**) zpravuje celkové žádosti, které jsou opět závislé na směru přenosu.
  - **OUT**: Pokud hostitel zaslal IN token(y) během datové fáze, musí potvrdit úspěšné přijetí dat. To je provedeno zasláním paketu OUT následovaného datovým paketem nulové délky. Pokud zařízení obdrží nulový paket a potvrdí jej (ACK), je stavová fáze kompletní a zařízení připraveno pro interpretaci dalšího příkazu. V případě chyby je opět zaslán paket STALL. Pokud zařízení nestihlo zpracovat dané informace, přichází paket NAK indikující hostiteli, aby stavovou fázi opakoval později.
  - **IN**: Pokud hostitel zaslal OUT token(y) během datové fáze, musí zařízení potvrdit úspěšný příjem dat zasláním prázdného datového paketu jako odpověď na IN paket odeslaný hostitelem. Token ACK, STALL a NAK mají stejný význam jako v předchozím případě.

1.3.5.2 Přerušované toky dat (Interrupt Transfers)

Ve většině zařízení se s pojmem přerušení pojí žádost o přerušení, která je vygenerována hardwarem. Naproti tomu u USB technologie pokud zařízení potřebuje získat pozornost hostitele, musí počkat dokud jej hostitel nevyzve k činnosti. I přes tento fakt mají přerušované toky jisté vlastnosti, které musí být dodrženy. Patří mezi ně:

- Garantované zpoždění.
- Využití proudových rour – jednosměrně.
- Detekce chyb.

Jsou typicky neperiodické a žádosti o přerušení ukládají do front, dokud si hostitel nevyžádá od zařízení zaslání dat. Velikost maximální užitečné informace je 8 bytů u low-speed zařízení, 64 bytů u zařízení full-speed a 1024 bytů u zařízení pracujících s high-speed rychlostmi. Varianty přerušovaných toků dat jsou tyto:

- **IN**: Hostitel periodicky vyzývá koncový bod. Doba tohoto vyzývání je specifikována deskriptorem koncového zařízení. Každá takováto výzva zahrnuje zaslání tokenu IN. Při ztrátě tohoto tokenu je paket ignorován. Pokud zařízení řadilo žádosti o přerušení do fronty, odešle datový paket obsahující data týkající se přerušení vzniklého při přejetí tokenu IN. Hostitel potvrdí úspěšné přijetí dat pomocí ACK, zatímco při porušení dat nereaguje. V případě, že nebylo vygenerováno žádné přerušení a přišlo vyzvání od hostitele, zařízení odpoví zasláním NAK. Dorazí-li paket poškozen, zařízení odešle paket STALL a tím požádá hostitele o opětovné zaslání IN paketu.
- **OUT**: Jestliže chce hostitel odeslat zařízení přerušovaný tok dat, vyžaduje to odeslání paketu OUT následovaného těmito daty. Pokud dojde k poškození jednoho z paketů, zařízení ignoruje oba. Je-li koncový bod prázdný a data byla v pořádku doručena, je zasláno potvrzení o úspěšném doručení (ACK). Pokud by koncový bod prázdný nebyl, bylo by to oznámeno prostřednictvím NAK. V případě chyby pak opět přichází na řadu paket STALL.

1.3.5.3 Izochronní toky dat (Isochronous Transfers)

Tyto přenosy se vykonávají průběžně a periodicky. Většinou obsahují časově citlivou informaci jako audio nebo video stream, pro kterou je zpoždění nepřípustné. Jsou však povoleny ztráty.

Izochronní přenosy garantují šířku přenášeného pásma a omezené zpoždění. Využívají jednosměrné proudové roury, detekují chyby prostřednictvím CRC a nezaručují doručení paketu. Využívají se pouze u full-speed a high-speed módů.

Velikost maximální užitečné informace je specifikována deskriptorem koncového bodu a může nabývat až 1024 bytů. Tato velikost má značný vliv na šířku přenosu pásma kladenou na sběrnici. Takto zasílaná data můžou měnit svou délku s každou transakcí. Jak již vyplývá z výše popsaného, izochronní přenosy nepodporují ustanovení spojení ani potvrzování paketů.

1.3.5.4 Nárazové toky dat (Bulk Transfers)

Nárazové toky nacházejí své uplatnění při přenosu velkého množství dat. Příkladem můžu být odeslání dokumentu tiskárně. Korekce chyb je řešena prostřednictvím pole CRC v datovém paketu a pomocí mechanismu detekce/přeposílání.

Nárazové přenosy šetří provoz sběrnice. Jestliže je sběrnice zaneprázdněna jinými typy přenosů, nárazová data využívají minimální šířku sběrnice. Ve výsledku jsou tedy používány pro

časově nezávislou komunikaci bez garance zpoždění. Jako u izochronních přenosů je zde využíváno proudových rour a full-speed a high-speed módů.

1.4 USB deskriptory

Všechny USB zařízení mají hierarchii popisovačů (deskriptorů), které popisují hostiteli zařízení, kdo jej vytvořil, jakou verzi standardu USB podporuje, jakými způsoby může být zařízení nakonfigurováno, počet koncových bodů atd. Mezi nejčastěji používané deskriptory patří:

- Deskriptor zařízení (Device Descriptor)
- Deskriptor konfigurace (Configuration Descriptor)
- Deskriptor rozhraní (Interface Descriptor)
- Deskriptor koncového bodu (Endpoint Descriptor)
- Deskriptor řetězce (String Descriptor)

USB zařízení mohou mít pouze jeden deskriptor zařízení. Ten obsahuje informace o tom, kolik možných konfigurací dané zařízení podporuje nebo např. Product a Vendor ID, sloužící pro načtení příslušného ovladače.

Konfigurační deskriptor specifikuje například zda-li je zařízení napájeno ze sběrnice, má-li vlastní zdroj napájení nebo počet rozhraní zařízení. Při inicializaci hostitel načte deskriptor zařízení a rozhodne se, kterou konfiguraci použít (může být použita pouze jedna). Změna této konfigurace vyžaduje ukončení aktivit na všech koncových bodech.

Deskriptor rozhraní je seskupení koncových bodů do funkční skupiny vykonávající operace charakteristické pro dané zařízení. Pro osvětlení tohoto pojmu si lze představit multifunkční zařízení (fax/skener/tiskárnu). Deskriptory rozhraní pak popisují koncové body potřebné pro funkci tohoto zařízení (deskriptor rozhraní č. 1 popisuje funkci faxu, deskriptor rozhraní č. 2 popisuje tiskárnu atd.). Na rozdíl od konfiguračního deskriptoru je možné pracovat s více deskriptory rozhraní současně.

Deskriptory koncového bodu jsou použity pro specifikaci typu přenosu a jeho směru, vyzývacího intervalu a maximální velikosti paketu. Nultý koncový bod je vždy využíván jako kontrolní koncový bod a jako takový nemá deskriptor.

Popis atributů jednotlivých deskriptorů je nad rámec této práce. Tento popis můžeme na jít v USB standardu [7] nebo v odkazované literatuře [4].

2 Popis architektury HC08

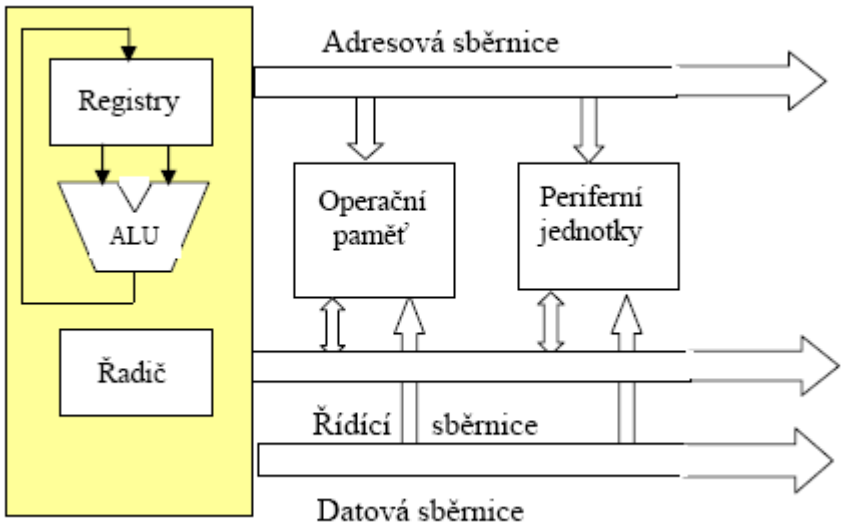
2.1 Vestavěné systémy

Všude okolo nás se vyskytují výpočetní systémy. Obvyklým představitelem je výpočetní počítač PC s volitelnými periferiemi. Typickým rysem takového PC je především jeho univerzálnost. Je tedy možné využít výpočetní výkon k různým úkonům.

Vestavěný systém je takový systém, který je součástí jiného systému a zpracování dat není viditelné uživateli. Tyto systémy jsou navrhovány pro konkrétní výpočetní úlohy. Příkladem využití vestavěného systému může být mobilní telefon, číslicový osciloskop nebo klávesnice počítače.

2.1.1 Mikroprocesor

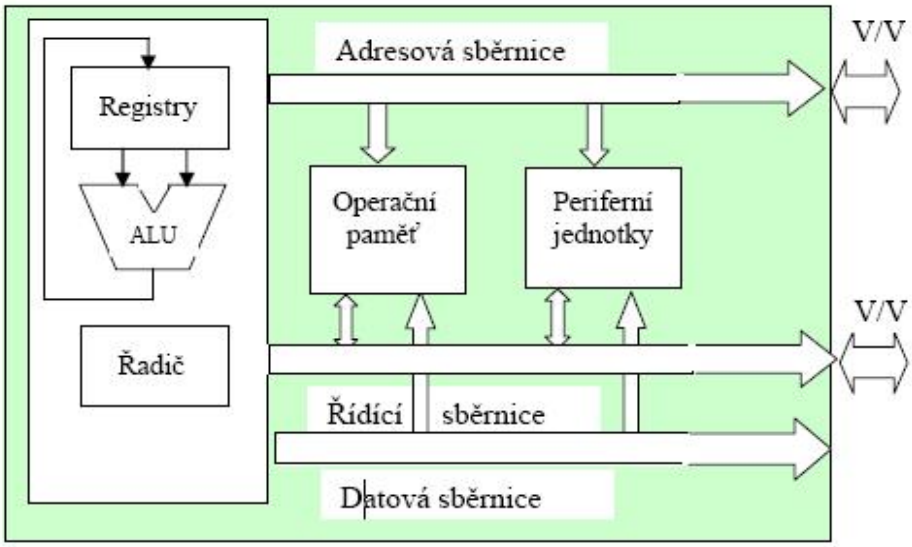
Mikroprocesor je tvořen aritmeticko-logickou jednotkou (ALU), instrukčním dekodérem, registry a řadičem. Chod mikroprocesoru je řízen řadičem. Řadič je tvořen registrem instrukcí, obvodem pro dekódování instrukce a řídicím obvodem, což souhrnně nazýváme řídicí částí. Aritmeticko-logická jednotka, soubor registrů – tzv. paměťová část a sběrnice tyto části propojují. Pro chod počítače musíme přidat vnější paměť a další periferní jednotky. Struktura mikroprocesoru je znázorněna na obrázku 2.1.



Obrázek 2.1: Struktura mikroprocesoru

2.1.2 Mikrokontrolér

Díky svým miniaturním rozměrům je možné mikrokontroléry použít všude tam, kde jsou vyžadovány operace se složitou logikou a komunikace s uživatelem. Mezi nejznámější výrobce současné doby patří Atmel, Motorola, Siemens a další. Struktura mikrokontroléru je znázorněna na obrázku č. 2.2.



Obrázek 2.2: Struktura mikrokontroléru

Mikrokontorléry na rozdíl od mikroprocesorů mají operační paměť i periferní jednotky umístěny přímo na čipu. Jádro mikrokontroléru zůstává většinou stejné, odlišuje se pouze jednotlivými periferiemi a velikostí paměti. S ostatními periferními zařízeními komunikují prostřednictvím vstupně-výstupního rozhraní. Důležitým rysem je také šířka sběrnice.

2.1.3 Architektura CISC a RISC

Rodiny mikrokontrolérů (občas i některé typy) se od sebe liší sadou instrukcí, kterou disponují. Podle vlastností instrukční sady můžeme tedy mikrokontroléry rozdělit na dvě kategorie: CISC a RISC.

2.1.3.1 CISC

Tato architektura byla používána na počátku, kdy paměti byl nedostatek a její cena byla vysoká.

Mezi základní vlastnosti této architektury patří velké množství instrukcí, velké množství adresovacích módů, proměnná délka instrukcí a složité adresování. Architektura efektivně využívá paměť. Typickým představitelem je např. Motorola.

Rostoucí sada instrukcí však činila problémy při překladu. Z tohoto faktu vyplývá, že je vhodnější použít větší množství instrukcí jednodušších, jak je tomu právě u architektury RISC.

2.1.3.2 RISC

Jak již bylo řečeno výše, architektura používá jednouché efektivní instrukce. Byla navržena podle reálných potřeb při programování mikrokontorlérů.

Mezi její základní vlastnosti patři redukovaná sada instrukcí s menší složitostí, instrukce mající pevnou délku, použití dvou základních instrukcí pro přístup do paměti (LOAD a STORE), velké množství registrů a jednoduché adresování. Nevýhodou této architektury je častější čtení instrukcí z paměti.

2.2 Platforma HC08

Pro implementaci USB klíče jsem zvolil mikrokontrolér rodiny Motorola MC68HC908JB8 [2]. Hlavním důvodem byla přítomnost modulu pro USB komunikaci (verze 1.1 low-speed), který je implementován na čipu daného mikrokontroléru.

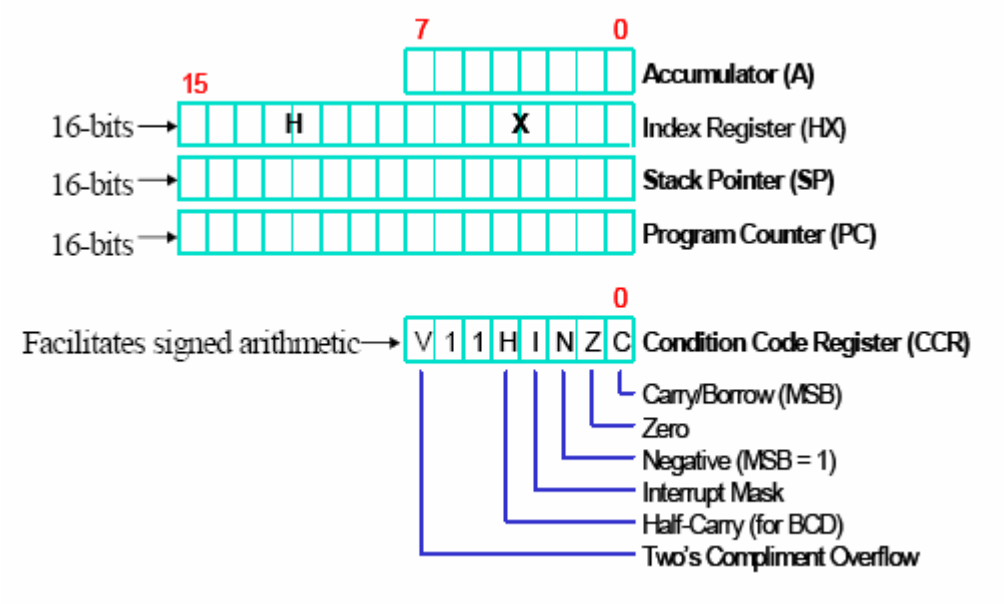
Rodina mikrokontrolérů HC08 je postavena na jádře CPU08 [6]. Mezi typické vlastnosti tohoto jádra patří omezený počet registrů (akumulátor, index-registr, zásobník, ukazatel instrukcí a příznakový registr), rychlý přístup do paměti a velké množství adresových módů. Významná je také podpora jednotného adresového prostoru pro paměť dat, programu i I/O periferií. CPU využívá von Neumannovu architekturu, hardwarovou podporu násobení a dělení a také podporu řízení spotřeby.

Další ze zajímavostí rodiny HC08 je FLASH paměť, která umožňuje naprogramování mikrokontroléru bez použití programátoru. S touto pamětí může pracovat i uživatelský program.

2.2.1 Programovací model

Programovací model je ve skutečnosti skupina registrů, bez kterých se programátor neobejde. Mimo tento model je nutné znát i instrukční sadu.

Programový model je znázorněn na obrázku č. 2.3. Tyto registry nejsou součástí paměťového prostoru.



Obrázek 2.3: Programovací model

Popis registrů:

- A - 8bitový akumulátor pro obecné použití. Často využíván pro uložení výsledků aritmetických operací.
- HX - Index registr HX je tvořen spojením dvou osmibitových registrů H a X. Umožňuje adresovat prostor o velikosti 64 KB.
- SP - 16bitový ukazatel zásobníku. Uchová pozici prvního volného místa v zásobníku. Po resetu mikrokontroléru ukazuje na adresu 0x00FF (vrchol zásobníku). Při vložení dat do zásobníku se SP snižuje, při výběru dat je tomu naopak. Pomocí zásobníku se předávají parametry podprogramu.
- PC - Programový čítač obsahující adresu příští instrukce. Standardně je automaticky inkrementován. Při volání podprogramu, skoku či přerušení je nastaven podle kontextu. Po resetu obsahuje resetovací vektor uložený na adrese 0xFFFE a 0xFFFF.
- CCR - 8bitový stavový registr obsahují globální masku přerušení a příznaky, které indikují výsledek právě prováděné instrukce.

V - příznak přetečení - Nastaven při přetečení výsledku v doplňkovém kódu. Tento příznak je důležitý pro znaménkovou aritmetiku.

H - příznak poloviční přenosu - Nastaven při přenosu mezi třetím a čtvrtým bitem akumulátoru během operací ADD a ADC. Využívá se při aritmetických operacích v BCD kódu.

I - maska přerušení - Slouží ke globálnímu povolení přerušení. Je-li tento příznak nastaven, žádná žádost o přerušení se neuplatní. Bit je automaticky nastaven při výskytu přerušení.

N - příznak záporného výsledku - Nastaven pokud při aritmetické, logické nebo přesouvací operaci je výsledek záporný.

Z - příznak nulového výsledku - Nastaven pokud při aritmetické, logické nebo přesouvací operaci je výsledek nula.

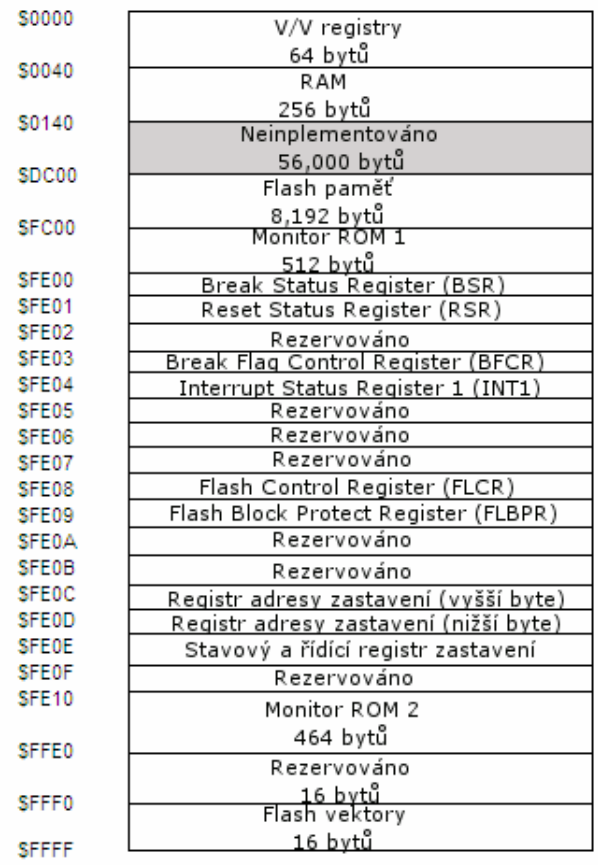
C - přenos/výpůjčka - Nastaven pokud při operaci součtu dojde k přenosu ze sedmého bitu nebo operaci rozdílu k výpůjčce. Příznak je také ovlivňován dalšími instrukcemi např. instrukcemi skoku, posuvy a rotacemi.

2.2.2 Paměťový model

Mikroprocesory HC08 mohou adresovat prostor o velikosti 64 kB. Organizace paměti je znázorněna na obrázku 2.4. Organizace paměťového prostoru se u různých druhů mikrokontrolérů liší. Při přístupu do neimplementované oblasti paměťového prostoru může dojít k resetu.

2.2.3 Instrukční sada

Mikrokontroléry rodiny Motorola obsahují instrukční sadu CISC a jejich instrukce lze rozdělit do několika skupin. S instrukcemi se také pojí adresové módy. Kompletní seznam instrukcí a popis adresových módů je možné nalézt v literatuře [6].



Obrázek 2.4: Mapa paměti MC68HC908JB8

1. **Instrukce pro přesuny dat** - Instrukce patřící do této skupiny slouží k přesunům dat mezi pamětí/registry a registry HC08. K typickým instrukcím z této třídy patří instrukce LDA (LDX, LDHX) resp. STA (STX, STHX) sloužící k načtení/zapsání hodnoty do/z akumulátoru. Jako další instrukce z této skupiny zmiňme instrukce pro práci se zásobníkem. Patří sem instrukce pro vložení dat na zásobník (PSHA, PSHX, PSHH) a instrukce pro vyjmutí dat ze zásobníku (PULA, PULX, PULH).
2. **Aritmetické instrukce** – Aritmetické instrukce provádějí základní operace s daty. K typickým instrukcím patří ADD ,SUB, MUL a DIV. Dále INCA a DECA pro inkrementaci a dekrementaci akumulátoru. Nebo instrukce pro porovnání obsahu akumulátoru s danou hodnotou (CMP).
3. **Logické instrukce** - Logické instrukce umožňují programátorovi pracovat s konkrétními bity daných bytů. Z pohledu těchto instrukcí můžeme na byte pohlížet ne jako na 8bitovou číselnou hodnotu, ale jako na jednorozměrné pole bitů, v němž jsou bity uloženy tak, že nejvíce významný (nejlevější) bit bytu je na indexu 7 a nejméně významný (nejpravější) bit bytu je na indexu 0. Patří sem instrukce pro provádění logických operací (AND, ORA), instrukce pro práci s jednotlivými bity (BCLR, BSET) a instrukce pro rotaci a aritmetický posuv (ASLA, ROLA).

4. **Řídící instrukce** – Tyto instrukce ovlivňují běh celého programu a umožňují jeho větvení a rozdělování do logických celků. Patří sem instrukce pro provádění skoků (BRA, BGE, BRCLR).

2.2.4 Podsystém přerušení

Podsystém přerušení umožňuje vykonat obslužný podprogram asynchronně vůči běhu hlavního programu. Pro ucelenou představu o přerušení je nejprve potřeba popsat význam podprogramu. V případě obou je potřeba uložit na zásobník údaje, které umožní návrat do místa volání.

2.2.4.1 Podprogram

Pro efektivní práci s programovacími jazyky se zavedly různé metodiky postupů při tvorbě programů. Jedním z těchto postupů je dekompozice složitého problému na několik problémů jednodušších. Typickým příkladem je rozdělení programu na podprogramy. Podprogram je kód začínající návěštím (v moderních programovacích jazycích se jedná o jeho název) a končící návratem z podprogramu. V architektuře HC08 je využito instrukcí BSR nebo JSR pro volání podprogramu a instrukce RTS pro návrat z podprogramu. Volání podprogramu je tedy předem dáno a je prováděno synchronně vzhledem k hlavnímu toku dat.

2.2.4.2 Obsluha přerušení

Přerušení je na rozdíl od podprogramu vykonáno asynchronně vzhledem k hlavnímu toku programu. Obsluha přerušení je reakce na událost vzniklou při běhu programu. Tyto události se vyhodnocují, za definovaných podmínek jsou akceptovány a řízení je předáno obslužné rutině pro dané přerušení. Obslužné rutiny jsou definovány prostřednictvím tzv. vektorů přerušení. Jedná se v podstatě o tabulku umístěnou na konci adresového prostoru obsahující adresy obslužných rutin.

Přerušení mohou vznikat jak na základě událostí externích (RESET, náběžná hrana), tak na základě událostí interních generovaných instrukcí nebo speciálním stavem. Všechny přerušení je pak možné maskovat.

2.2.5 Implementace USB rozhraní na platformě HC08

Tato pasáž se zabývá popisem USB registrů řízení, kontroly a registry datovými. Jsou zde popsány pouze nejzákladnější vlastnosti pro vytvoření představy o náročnosti ovládání. Pro podrobnější údaje o těchto registrech, technologiích a hardwarovém provedení doporučuji nahlédnout do datasheetu některého z mikrokontrolérů (viz. literatura [2]). Následující registry řídí a monitorují operace dané specifikací USB 1.1:

- USB address register (UADDR)
- USB control registers 0-4 (UCR0-UCR4)
- USB status registers 0-1 (USR0-USR1)
- USB interrupt registers 0-2 (UIR0-UIR2)
- USB endpoint 0 data registers 0-7 (UE0D0-UE0D7)
- USB endpoint 1 data registers 0-7 (UE1D0-UE1D7)
- USB endpoint 2 data registers 0-7 (UE2D0-UE2D7)

2.2.5.1 Popis jednotlivých registrů

- **Adresový registr** - Jedná se o základní registr USB rozhraní v mikrokontroléru. Obsahuje bit USBEN, pomocí kterého se povoluje, případně zakazuje USB rozhraní. Dále následuje šest bitů UADD pro uchování adresy zařízení po konfiguraci hostitelem.
- **Kontrolní registry** - USB rozhraní je vybaveno čtyřmi kontrolními registry. V těchto registrech se nastavují vlastnosti koncových bodů a typy dat. Obsahují údaje o velikosti dat odesílaných, možnost nastavení přechodu do režimu nízké spotřeby (suspend mode), využití interního pull-up rezistoru a nastavení zasílání STALL paketů na vybraném koncovém bodě.
- **Stavové registry** - Rozhraní je vybaveno dvěma registry, které obsahují informace o typu zpracovávaného tokenu, velikosti příchozích dat a o stavových paketech.
- **Registry přerušení** – Jedná se o tři registry obsahující povolení přijímat a odesílat data z jednotlivých koncových bodů, identifikaci konce paketu, identifikaci signálu reset od hostitele a příznaky informující o vyprázdnění a naplnění datových registrů.
- **Datové registry** – Rozhraní obsahuje 8 datových registrů a to pro každý koncový bod (zvlášť příchozí a odchozí). Umožňuje tedy přenášet data po 8 bytech.

2.2.5.2 Přerušení v módu nízké spotřeby

USB modul generuje přerušení do CPU nachází-li se v režimu nízké spotřeby a na sběrnici byla detekována žádost o návrat do aktivního stavu. Tato událost nastaví bit RESUMF v registru přerušení UIR1.

2.2.5.3 Přerušení identifikující konec paketu

USB modul generuje přerušení (End-of-Packet Interrupt) detekuje-li na datových vodičích konec paketu. USB modul nastaví bit EOPF a generuje přerušení v případě, že je nastaven bit EOPIE v registru přerušení UIR0.

3 Návrh USB klíče

Návrh hardwaru, který pracuje s USB rozhraním, je na první pohled velmi obtížný. Na tomto faktu má podíl především rozsáhlá USB specifikace, jenž je popsána na několika stovkách stránek. Navrhujeme-li zařízení podléhající třídnímu standardu (USB Class Standard), např. zařízení HID (Human Interface Device), je vyžadována znalost této třídy a požadavků na implementaci. Při návrhu hostitele máme na výběr ze tří různých standardů. Žádný z výše zmíněných příkladů není v USB specifikaci obsažen a je tedy nutné studovat velké množství zdrojů.

Dalším důležitým požadavkem je připojení a komunikace s hostitelským počítačem. Tato komunikace se děje prostřednictvím ovladače operačního systému. Každé USB zařízení obsahuje VID a PID umožňující operačnímu systému nalézt příslušný ovladač.

Při tvorbě hardwaru máme tedy dvě možnosti. První z nich je napsat vlastní ovladač pro vytvářené USB zařízení. To je možné s využitím Microsoft SDK (Software Development Kit) a DDK (Driver Development Kit) [16]. Tyto volně dosažitelné balíky obsahují potřebné hlavičkové soubory, dokumentace k jednotlivým funkcím a příklady ovladačů. Toto programování však vyžaduje podstatné znalosti pracovních principů a struktury daného operačního systému a jeho ovladačů.

Druhou možností je využít standardního ovladače Windows pro Human Interface zařízení (HID). Máme tedy možnost zamaskovat potřebná data jako HID paket a využít tak tohoto ovladače. Hlavního výhoda spočívá v tom, že ovladač již nemusíme vyvíjet a je součástí každého operačního systému Windows. Můžeme se tedy soustředit na naprogramování vlastní aplikace. I přes tyto zjevné klady existují i záporné vlastnosti tohoto postupu. Jednou z nich je složitost ošetření situací na straně firmwaru. Ten musí splňovat požadavky kladené na HID zařízení [5]. Druhým a závažnějším problémem je možnost změny USB ovladače s každou další novou verzí operačního systému nebo aktualizací jeho komponent. Z tohoto důvodu je nutné provést více testů funkčnosti pro spolehlivý běh zařízení.

Stejné postupy nám nabízejí i jiné operační systémy. Můžeme tedy využít již vytvořeného ovladače pro HID zařízení nebo napsat svůj vlastní. V případě tvorby vlastního USB ovladače pro operační systém Linux, bych rád odkázal na knihu Linux Device Drivers, která je dostupná online a věnuje celou kapitolu 13 tvorbě USB ovladačů (viz. literatura [15]).

V této práci jsem se rozhodl využít již existujících ovladačů a při návrhu USB klíče využít komunikaci prostřednictvím rozhraní HID.

3.1 Rozhraní HID

USB architektura dala personálním počítačům schopnost připojit různorodá zařízení prostřednictvím jednoduchého čtyřvodičového kabelu. Protokoly této architektury umožňují periferie nakonfigurovat

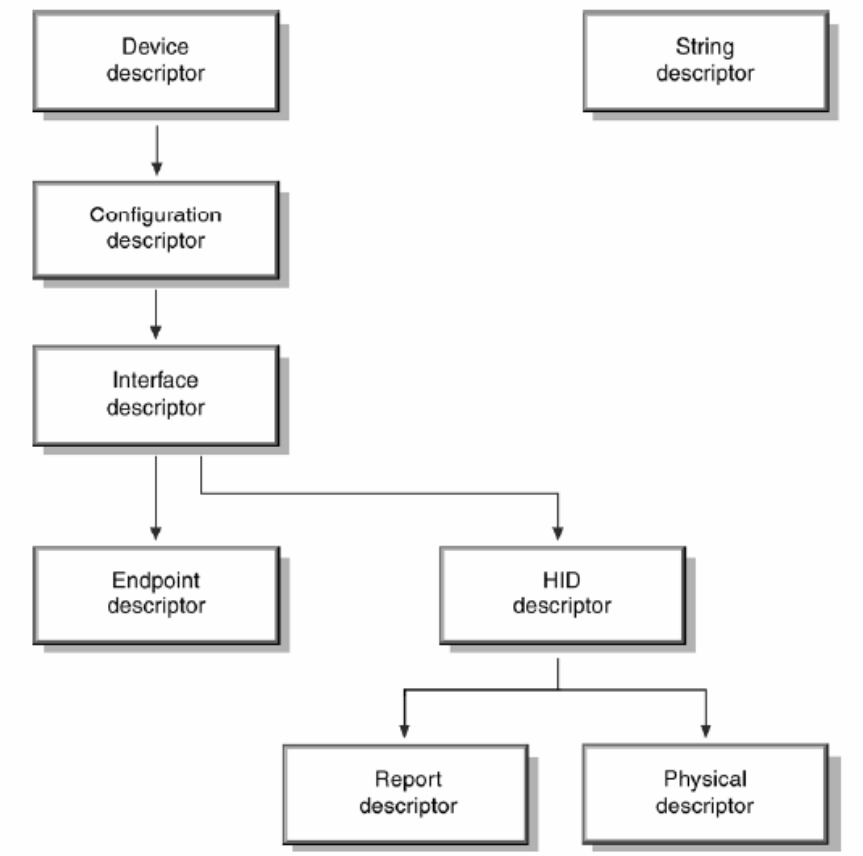
při startu počítače nebo za jeho běhu. Připojovaný hardware můžeme rozdělit do několika různých tříd, z nichž každá definuje obvyklé chování a protokoly, pro tato zařízení typická. Mezi nejčastěji používané třídy patří Mass storage (pro pevné disky), Display (monitory) nebo Human Interface.

Třída HID popisuje zařízení, která jsou využívaná uživatelem pro obsluhu počítačového systému. Typickými představiteli jsou tedy klávesnice a myši, ale patří sem i přepínače a herní volanty a pedály. Tato třída se velmi často využívá pro periferie, které nevyžadují interakci od uživatele, ale používají stejný formát dat.

Informace o USB zařízení, uložené v paměti ROM, slouží pro popis chování. Tyto informace se nazývají deskriptory a byly popsány v kapitole 1.4. Jestliže zařízení spadá pod tuto konkrétní třídu, je potřeba přidat ještě další deskriptory pro jeho popis. Jedná se o HID deskriptor a Report deskriptor, na který HID deskriptor odkazuje. Report deskriptor popisuje data, která hardware generuje a jejich význam. Obrázek 3.1 znázorňuje hierarchii takovýchto deskriptorů. Podrobné informace o deskriptorech jsou uvedeny v literatuře [5].

USB zařízení jsou rozděleny do tříd, které mají stejné protokoly a využívají příslušný ovladač. Zařízení může patřit do jedné nebo více tříd. Příkladem může být telefon, který přísluší do tříd HID, telephony a audio. Je to umožněno díky faktu, že třída je specifikována v deskriptoru rozhraní. Hostitel získá informaci o příslušné třídě prostřednictvím položky *bInterfaceClass*, která je obsažena v deskriptoru zařízení. Pro HID zařízení je hodnota této položky rovna 3.

Členové této třídy komunikují s ovladačem prostřednictvím kontrolní a přerušovací roury. Zatímco kontrolní roura je využita pro přijímaní a odesílání žádostí, přenos dat vyžádaný ovladačem a příjem dat od hostitele, přerušovací je použita pro asynchronní přenos dat.



Obrázek 3.1: Hierarchie deskriptorů

3.2 Vlastnosti hardwarových klíčů

Pod pojmem USB klíč si můžeme představit velké množství zařízení. Tímto pojmem můžeme rozumět klasické USB zařízení s externí pamětí pro ukládání dat, dále pak zařízení obsahující šifrovací klíč pro zabezpečení dat nebo hardwarový klíč pro spouštění aplikací popř. uzamykání počítače. Pro všechny výše zmíněné varianty existují společné charakteristické vlastnosti. Mezi ty základní patří:

- Malá velikost a hmotnost
- Paměť pro data
- Napájení ze sběrnice
- Rychlý datový přenos

Z výše vyjmenovaných periferií jsem zvolil variantu hardwarového klíče pro zabezpečování aplikací. Aplikace je tedy závislá na hardwarovém prvku a její zabezpečení je na znatelně vyšší úrovni než při využití klíče softwarového. Popisu implementace se věnuje kapitola 4.

3.3 Architektura USB klíče

USB klíč je navržen jako zařízení řadící se do třídy HID. Neslouží pro interakci s uživatelem, ale využívá obdobný datový formát. Jako zařízení spadající do této kategorie musí splňovat dané požadavky, mezi které patří např. přítomnost HID a report deskriptoru nebo schopnost odpovídat na standardní žádosti pro tuto třídu specifické. Po připojení k PC je zařízení identifikováno jako „xslima00 security key“. USB klíči je přiřazeno VendorID 0x0425 (identifikační číslo společnosti Motorola) a ProductID 0xC120 (náhodné číslo produktu). USB klíč ve své paměti ROM uchovává osmibitový předdefinovaný klíč, který je neměnný a nemazatelný. Prostřednictvím softwaru uloženého v PC můžeme odesláním standardní žádosti tento klíč získat a přesvědčit se tak o přítomnosti zařízení. Architektura navrženého hardwarového klíče je zobrazena na obrázku č. 3.2.

Následující tabulka 3.1 popisuje základní žádosti implementované v hardwarovém klíči. Jelikož pro samotnou funkčnost klíče není zapotřebí využívat jiného koncového bodu než-li nultého a zařízení obsahuje pouze jedno konfigurační nastavení, nejsou implementovány všechny obsluhy žádostí, které by v běžném zařízení standartu HID musely být přítomny. Význam žádostí a popis jejich parametrů je obsažen v literatuře [4] a [5] nebo na www stránkách [7].

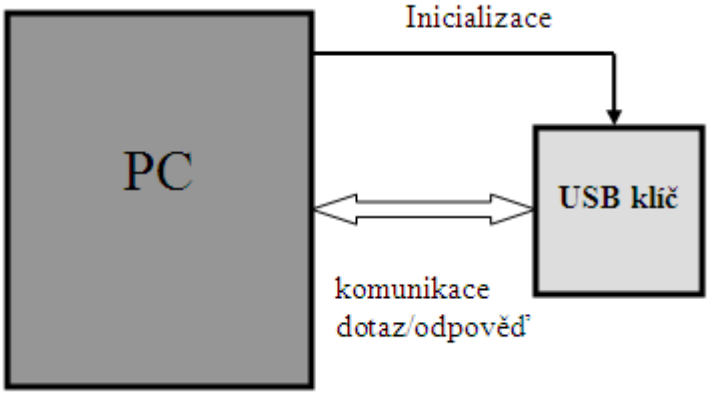
3.4 Hardwarový návrh

Tato kapitola popisuje hardwarový návrh USB klíče a jednotlivé součástky použité ve schématu zapojení. Na obrázku 3.3 je znázorněno schéma hardwarového klíče. Návrhy plošných spojů jsou připojeny v příloze B.

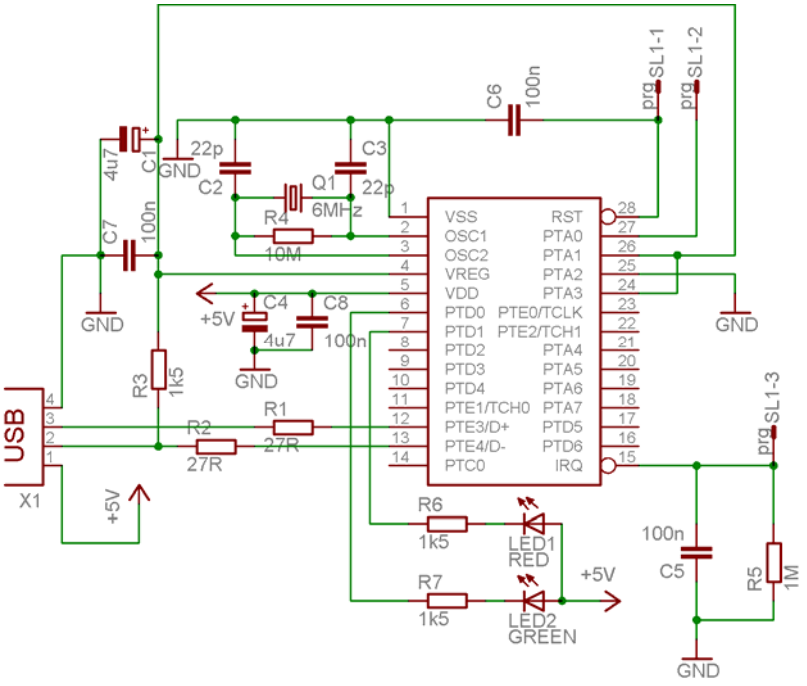
Ze schématu je patrné, že základem návrhu je mikrokontrolér, který tvoří základ celého USB klíče. Aby byla zajištěna správná funkčnost mikrokontroléru, je potřeba využít několika externích součástek, o jejichž typu a hodnotách se můžeme dočíst v datasheetu [2].

bmRequestType	bRequest	wValue	wIndex	wLength	Data
1000 0000b	GET_STATUS (0x00)	Zero	Zero	Two	Device Status
1000 0001b	GET_STATUS (0x00)	Zero	Interface	Two	Interface Status
1000 0010b	GET_STATUS (0x00)	Zero	Endpoint	Two	Endpoint Status
0000 0000b	CLEAR_FEATURE (0x01)	Feature Selector	Zero	Zero	None
0000 0001b	CLEAR_FEATURE (0x01)	Feature Selector	Interface	Zero	None
0000 0010b	CLEAR_FEATURE (0x01)	Feature Selector	Endpoint	Zero	None
0000 0000b	SET_FEATURE (0x03)	Feature Selector	Zero	Zero	None
0000 0001b	SET_FEATURE (0x03)	Feature Selector	Interface	Zero	None
0000 0010b	SET_FEATURE (0x03)	Feature Selector	Endpoint	Zero	None
0000 0000b	SET_ADDRESS (0x05)	Device Address	Zero	Zero	None
1000 0000b	GET_DESCRIPTOR (0x06)	Descriptor Type & Index	Zero or language ID	Descriptor length	Descriptor
1000 0000b	GET_CONFIGURATION (0x08)	Zero	Zero	1	Configuration value
0000 0000b	SET_CONFIGURATION (0x09)	Configuration Value	Zero	Zero	None
1000 0001b	GET_INTERFACE (0x0A)	Zero	Interface	One	Alternace Interface
0010 0001b	SET_PROTOCOL (0x0B)	0/1	Interface	Zero	None
1010 0001b	GET_PROTOCOL (0x03)	0	Interface	1	0/1
1010 0001b	GET_REPORT (0x01)	2	Device	8	KeyID

Tabulka 3.1: Standardní žádosti implementované v USB klíči



Obrázek 3.2: Architektura USB klíče



Obrázek 3.3: Schéma zapojení USB klíče

3.4.1 Popis součástek

Abychom zabránili problémům s šumem, jsou mezi napájecí a zemnící vývody mikrokontroléru připojeny kondenzátory C4, a elektrolytický kondenzátor C8. Mikrokontrolér je vybaven vlastním napěťovým regulátorem V_{REG}. Ten je použit pro interní operace MCU a řízení USB dat. Tento napěťový regulátor vyžaduje dva kondenzátory (C1 a C7) o velikosti 4,7 µF a 0,1 µF. V případě programování na čipu je dále nutné připojit externí krystal na vývody OSC1 a OSC2. Frekvenční

rozsah krystalu a dalších externích součástek pro naprogramování čipu je opět možné vyčíst z datasheetu [2].

K samotnému programování na čipu je ještě potřeba využít tří základních vývodů mikrokontroléru. Tyto vývody jsou vyvedeny prostřednictvím slotů a tvoří komunikační rozhraní mezi zařízením a okolím. Zároveň je nutné, aby programovaný čip sdílel společnou zem spolu s programátorem. Samotná komunikace pak probíhá sériově pouze po jednom vodiči (PTA0). Režim programování se nastaví prostřednictvím pinů PTA1-PTA3, na které se přivede logická „1“ nebo „0“ v závislosti na režimu programování.

Pro aplikační funkčnost USB klíče je zde dále připojen rezistor R3, který slouží jako externí USB pull-up rezistor. Pro ladění a signalizaci jsou využity LED diody D1 a D2.

3.4.2 Programování mikrokontroléru

Samotné programování mikrokontroléru je prováděno prostřednictvím vývojového prostředí softwaru Codewarrior, které nabízí firma Freescale. Tento produkt velmi usnadní práci při ladění zařízení, díky vestavěnému rozhraní pro sledování paměti, proměnných a obslužných rutin. Umožňuje také plnou simulaci v prostředí PC. Obsahuje editor zdrojového kódu se zvýrazněním syntaxe, práci s projekty a propracovanou nápovědu. Toto vývojové prostředí umožňuje vývoj aplikace prostřednictvím standardního assembleru nebo některého z vyšších programovacích jazyků (C/C++). Pro implementaci firmwaru USB klíče jsem zvolil assembler především kvůli jeho efektivnosti a kontrole nad každým bitem. Codewarirror není jediným řešením. Je možné využít jiných softwarových nástrojů. Popis instrukcí pro programování paměti je opět možno nalézt v datasheetu [2].

Jako hardwarový nástroj, spolupracující se softwarem Codewarrior, lze využít kit Janus [9]. Tento kit se specializuje na programování mikrokontrolérů NITRON, ale obsahuje i rozhraní MiniMON pro programování a ladění libovolného mikrokontroléru jádra HC08. Rozhraní umožňuje využít napájení aplikace z kitu i napájení kitu z aplikace, externí hodinový signál pro laděný mikrokontrolér o frekvenci 9,8304 MHz, logické signály a komunikační signál, prostřednictvím kterého probíhá celá komunikace. Další možností, jak naprogramovat mikrokontroléry jádra HC08, je MON08 MULTILINK kabel umožňující propojení pomocí standardního paralelního portu počítače.

4 Implementace

Tato kapitola popisuje implementaci firmwaru, který řídí mikrokontrolér komunikující s počítačem pomocí rozhraní USB verze 1.1.

4.1 Vývojové prostředky

Jako vývojový prostředek pro tvorbu firmwaru jsme zvolil software firmy Freescale, Codewarrior. Jeho výhody byly již popsány v kapitole 3.4.2 zabývající se návrhem zařízení. Pro tvorbu softwaru běžícího na cílové platformě Windows jsem se rozhodl využít Microsoft Visual Studia 2005. Tento pokročilý komerční nástroj nabízí mnohá vylepšení usnadňující práci. Současně s ním byl použit Windows Development Kit, který je zdarma ke stažení na stránkách firmy Microsoft [16] a obsahuje potřebné hlavičkové soubory a knihovny pro práci s jakýmkoliv hardwarem.

4.2 Firmware

Firmware uložený v USB klíči je pro snazší orientaci a přehlednost rozdělen do několika souborů.

1. **deskriptory USB zařízení** – Deskriptory USB zařízení umožňující hostiteli získat informace o nastavení a konfiguraci zařízení. Jsou umístěny v modulu `descriptors.h`.
2. **registry MC68HC908JB8** – Popis jednotlivých registrů a bitů spolu s adresami uložení v mikrokontroléru. Soubor byl vygenerován nástrojem Codewarrior 5.1. Modul má název `MC68HC908JB8.inc`.
3. **přerušovací rutina USB** – Modul popisuje obsluhu přerušení při příchodu žádosti o přerušení generovanou USB sběrnicí. Obsahuje zpracování jednotlivých paketů a specifických žádostí pro USB zařízení standardu HID. Tento modul je tvořen dvěma soubory `interrupt.asm` a `HID.asm`.
4. **hlavní modul** – Obsahuje definice proměnných a konstant, inicializaci zařízení, hlavní smyčku a definice přerušení. Je tvořen souborem `USBkey.asm`.

4.2.1 Konstanty a proměnné

V souboru `USBkey.asm` jsou uvedeny a okomentovány veškeré proměnné a konstanty potřebné pro běh aplikace. Pro přehlednost jsou zde uvedeny pouze klíčové proměnné. Pro snazší rozlišení jsou proměnné uvozeny pomocí písmena V a konstanty písmenem K.

Základem pro zpracování žádostí je fronta *Q_Setup*, která v sobě uchovává datový paket následující po SETUP tokenu. Fronta je tvořena několik proměnnými a má velikost 8 bytů. Do této fronty se ukládají data pro pozdější dekódování a zpracování.

Aby bylo možné určit v jaké fázi se ve zpracování příchozích žádostí nacházíme, byla použita proměnná *V_Transact*. Obdobně pro stav zařízení (zařízení připojeno, adresováno atd.) byly použity proměnné *V_Dev_State* a *V_Dev_Status*.

Jednou z nejdůležitějších konstant je *KeyIDPtr* obsahující adresu místa v paměti ROM. Prostřednictvím toho ukazatele je možné přečíst uložený klíč a odeslat jej.

Další důležitou konstantou je *K_FCPU* udávající velikost frekvence mikrokontroléru. Konstanta slouží k výpočtům využitých pro časovač.

4.2.2 Inicializace zařízení

Po připojení zařízení k USB sběrnici nebo po jeho resetování je nutné zařízení inicializovat. Děje se tak prostřednictvím skoku na návěští `_Startup`, v případě prvního připojení zařízení do systému, nebo skokem na návěští `Entry`, v případě resetu.

V první fázi je nastaveno umístění zásobníku na konec paměti RAM a konfigurační registr čipu tak, aby byla povolena instrukce STOP pro přechod do režimu nízké spotřeby, byl zakázán obvod COP a USB reset vyvolal resetování celého čipu. V této fázi je do stavového registru zařízení zapsán stav indikující napájení USB klíče (POWER) a jsou povoleny výstupní porty pro indikační diody.

V další fázi je zapotřebí nastavit časovač, který slouží pro měření doby neaktivity na sběrnici. U časovače není vyžadováno povolení přerušení. Je však využito interní předděličky v režimu 1:4. Pro rozpoznání neaktivity na sběrnici je využito kanálu časovače 0, který je nastaven v režimu výstupní komparace. Neobdrží-li zařízení po stanovenou dobu žádný signál od sběrnice USB a časovač dosáhne hodnoty, při které má dojít ke komparaci, je vyvoláno přerušení, které nastaví požadavek na přechod do režimu nízké spotřeby. Při jakékoliv aktivitě na sběrnici podprogram `Ref_TCH0` zajistí aktualizaci hodnoty, při které má dojít k přechodu do režimu nízké spotřeby přičtením konstanty k hodnotě uložené v registru kanálu 0 pro výstupní komparaci.

Nyní se zařízení nachází v počátečním stavu (DEFAULT). Dále je potřeba povolit USB modul, nastavit mu defaultní adresu, resetovat všechny příznaky, povolit přerušení pro identifikaci konce paketu, připravit zařízení pro příjem dat a inicializovat proměnné. Tyto požadavky zajistí podprogram `RST_USB`. Zařízení se nyní nachází v nečinném stavu (IDLE). Na závěr je potřeba zakázat přerušení, která nechceme využívat.

Konfigurace je dokončena a program vchází do hlavní smyčky. Jedná se v podstatě o nekonečnou smyčku, která neustále kontroluje, zda nenastala žádost na přechod do režimu nízké spotřeby. Jakákoliv aktivita na USB sběrnici vyvolá skok do modulu přerušení, kde jsou obsluhovány jednotlivé žádosti.

Následující příklad demonstruje výpočet hodnoty 3 ms, která je definována standardem pro přechod do režimu nízké spotřeby při neaktivitě na sběrnici. Výsledný počet cyklů se vždy přičte k aktuální hodnotě časovače, je-li zaznamenána aktivita na sběrnici.

$$1\_cyklus\_čítače = \frac{1}{\frac{f}{4}} = \frac{1}{\frac{3*10^6}{4}} = 1{,}3333\overline{3}\,\mu s \qquad (1)$$

$$pocet\_cyklu = \frac{t}{1\_cyklus\_čítače} = \frac{3*10^{-3}}{1{,}3333\overline{3}*10^{-6}} = 2250\,\text{cyklů} \qquad (2)$$

```
Ref_TCH0:

        ldhx    TCNTH           ; načti do H:X hodnotu časovače
        pshh                    ; H na zásobník
        clrh                    ; H = 0
        txa                     ; A = X
        add     #(K_Nch0 % 256) ; přičti k A konstantu
        tax                     ; ulož zpátky do X
        pula                    ; načti do A hodnotu ze zásobníku
        adc     #(K_Nch0 / 256) ; přičti konstantu
        sta     TCH0H           ; ulož do registru pro komparaci
        stx     TCH0L           ; stejně tak X
        bclr    TSC0_CH0F,TSC0  ; vynuluj příznak

RTCH0_EXIT:
        rts
```

4.2.3 Přerušovací rutina

Jak bylo uvedeno výše, hlavní program je prováděn v nekonečné smyčce a základní úlohu hraje rutina volaná při indikaci přerušení od modulu USB. Při skoku do rutiny nejprve kontrolujeme, zda-li poslední odeslaný paket nebyl STALL. Pokud ano, je potřeba, aby bylo zařízení resetováno. Jestliže bude zařízení neustále odesílat STALL pakety, USB sběrnice si tento reset vynutí. Také je kontrolováno, zda-li je nastaven příznak konce paketu. V takovémto případě je nutné aktualizovat časovač, aby nedošlo k přechodu do režimu nízké spotřeby.

```
USB_ISR:

    brclr   USR1_TXSTL,USR1,UISR_EOP          ; byl odeslán STALL paket při
                                              ; poslední transakci EP0 ?
    jsr     ECHO_STALL                        ; zařízení odpoví STALL,
                                              ; potřeba resetu

UISR_EOP:

    brclr   UIR1_EOPF,UIR1,UISR_EOP_END     ; detekován konec paketu ?
    bset    UIR2_EOPFR,UIR2                  ; vynuluj End-Of-Paket příznak
    bclr    b_Tsus,V_Timer                  ; vynuluj příznak suspend
                                            ; timeout
    jsr     Ref_TCH0                    ; refresh čítače, oddálení suspend

UISR_EOP_END:
```

Následuje kontrola příchozího a odchozího bufferu koncového bodu 0. Jestliže některý z těchto bufferů přijal SETUP token, jej následující data a odpověděl pomocí ACK, je tento stav indikován v registru UIR1. V dalším kroku rozpoznáme o jaký token se jednalo. V případě odchozího bufferu koncového bodu 0 je zřejmé, že se jedná o paket IN, příchozí buffer však může obsahovat token SETUP nebo OUT. Typ tokenu zjistíme podle registru USR0, jak je patrné z následujícího kódu.

```
     brset   USR0_SETUP,USR0,UISR_SETUP0     ; byl poslední token SETUP ?

UISR_OUT0:

    jsr     OUT_PKT                          ; obsluha OUT transakce
    bset    UIR2_RXD0FR,UIR2                ; připrav se na další OUT/SETUP
    bra     UISR_HID

UISR_SETUP0:

    jsr     SETUP_PKT                        ; obsluha SETUP transakce
    bset    UIR2_RXD0FR,UIR2                ; připrav se na další OUT/SETUP
    bra     UISR_HID
```

Jakmile je určen typ tokenu, následuje skok do podprogramu, který daný token zpracuje. Po rozpoznání jednotlivých paketů je volán podprogram `HID_Handler`, který zpracuje jednotlivé žádosti dané standardem.

4.2.3.1 Zpracování SETUP paketu

Na počátku je potřeba zjistit, zda přijatá data odpovídají našim požadavkům. Potřebujeme znát především typ dat a jejich velikost. Pokud data neodpovídají předpokladům, je SETUP paket zahozen.

```
   lda     USR0                    ; Status registr do A
   and     #%10001111              ; Vymaskování, typ dat a
                                   ; velikost paketu, RP0SIZx a R0SEQ
   cmp     #%00001000              ; Přijato 8 bytů typu DATA0?
   bne     SP_INVALID                 ; pokud ne, ignoruj
```

V opačném případě dochází k nakopírování přijatých dat do fronty pro zpracování a nastavení příznaku dokončení této fáze.

```
   ldx     #8

SP_LOOP:
   lda     (UE0D0-1),x
   sta     (Q_Setup-1),x
   dbnzx   SP_LOOP                 ; Zkopíruj UE0D[0..7] do Q_Setup[0..7]

   mov     #(1<<b_SETUP),V_Transact         ; SETUP fáze kompletní
```

Dále je třeba rozhodnout, jaký paket bude následovat. Tuto informaci můžeme zjistit z proměnné *V_bmReqType*, která je součástí dat uložených ve frontě *Q_Setup*. Pokud je následující paket IN, je zapotřebí nastavit, aby zařízení odesílalo pakety NAK pro případ, že by hostitel požadoval data dříve, než by byla zpracována. Pokud následuje paket OUT, je do stavové proměnné zařízení *V_Transact* zapsán příznak započetí datové fáze a připraven buffer pro přijetí dat. Vždy je nutné změnit typ datového paketu z DATA0 na DATA1. Jestliže další přenos dat nebude uskutečněn (např. žádost CLEAR_FEATURE), je nutné potvrdit přijatá data IN stavovou fází.

4.2.3.2 Zpracování OUT paketu

Stejně jako v případě zpracování SETUP paketu zjistíme z registru USR0 velikost přijatých dat. Pokud je velikost rovna nule, jedná se o stavový paket, kterým hostitel potvrdil přijetí dat. Tato událost způsobí blokování příchozích IN/OUT paketů a nastavení příznaku dokončení stavové fáze. Je totiž zřejmé, že další zpracovávaný paket bude SETUP.

Pokud je velikost přijatých dat různá od nuly, jedná se o typický OUT paket. V tomto případě jsou data z příchozích registrů UE0Dx nakopírována do příchozího bufferu.

```
TRX_LOOP:

        lda     UE0D0,x                 ; UE0Dx[V_UDR_Ptr]
                                        ; -> Q_Rx_Buf[V_Rx_Ptr]
        ldx     V_Rx_Ptr
        sta     Q_Rx_Buf,x

        inc     V_UDR_Ptr               ; inkrementuj
        inc     V_Rx_Ptr

        ldx     V_UDR_Ptr              ; if (X<V_UDR_Size),bra TRX_LOOP
        cpx     V_UDR_Size
        blo     TRX_LOOP
```

4.2.3.3 Zpracování IN paketu

Po vstupu do této obslužné rutiny zjišťujeme, zda byla započata datová či stavová fáze a podle tohoto faktu program větvíme. Při vstupu do stavové fáze nastavíme patřičné příznaky. Mimo to se zde nachází nastavení adresy USB zařízení. Toto nastavení smí být provedeno teprve po úspěšném potvrzení SET_ADRRESS žádosti zaslané hostitelem. Proto se nachází zde a ne v části zpracovávající žádosti hostitele.

Jedná-li se o typický IN paket, jsou do registrů UE0Dx nakopírována data, na které ukazuje pointer *V_Off_STA*. Data jsou posílána po osmi bytech. Proměnná *V_UDR_Size* uchová velikost právě odesílaného paketu, zatímco proměnná *V_Tx_Size* uchovává velikost celkovou. Na konci této obslužné rutiny je zapotřebí nastavit typ datového paketu, jeho velikost a povolit hostiteli vyzvednutí dat z koncového bodu.

```
lda     UCR0
and     #(1<<UCR0_TOSEQ)
eor     #(1<<UCR0_TOSEQ)    ; překlop PID DATA0/1
ora     V_UDR_Size          ; aktualizuj velikost přenášených dat
ora     #((1<<UCR0_TX0E)|(1<<UCR0_RX0E))    ; povol přenos
sta     UCR0
```

4.2.3.4 Zpracování žádostí

Po zpracování příchozích paketů následuje vždy skok do rutiny `HID_Handler`. Tato rutina má význam především pro zpracování SETUP paketů. Pro ostatní pakety pouze nastaví příslušné příznaky v pomocných proměnných. Jedná-li se však o SETUP paket, následuje skok do obslužné rutiny `PROC_SETUP`, která identifikuje a zpracuje žádost na USB zařízení podle standardu. Nejprve je nutné zjistit o jaký typ žádosti se jedná z proměnné *V_bmReqType*.

```
lda     V_bmReqType
and     #%01100000
                          ; D6..5 určuje typ
                          ;    0 - Standart
                          ;    1 - Class
                          ;    2 - Vendor
                          ;    3 - Reserved

                          ; Standard Device žádost ?
cmp     #%00000000
beq     PST_STANDARD
                          ; Class-specific žádost ?
cmp     #%00100000
beq     PST_CLASS
bra     PST_STALL         ; jinak STALL
```

Po zpracování typu žádosti je nutné zjistit o jakou žádost se jedná. K tomuto účelu je vyhrazena proměnná *V_bRequest*. Podle hodnoty obsažené v této proměnné provedeme skok do patřičné rutiny. Pro ilustraci je zde dále uvedena standardní rutina STD_GET_CONFIG, sloužící pro získání konfigurace zařízení.

Na počátku je opět nutné zjistit, zda-li je žádost bezchybná. V předcházejících krocích jsme poznali, že se jedná o standardní žádost s dotazem na konfiguraci zařízení. Jsou zde však i další proměnné (*V_wLength*,*V_wValue* a *V_wIndex*), které musí splňovat požadavky ze standardu plynoucí. Jak je vidět na následujícím kódu, je nutné aby všechny proměnné byly nulové mimo *V_wLength* udávající velikost odesílaných dat, která je nastavena na hodnotu jedna.

```
lda     V_wLength_H
ora     V_wValue_L
ora     V_wValue_H
ora     V_wIndex_L
ora     V_wIndex_H
bne     SGC_STALL
lda     V_wLength_L             ; velikost != 1 byte ?
cmp      #1
bne      SGC_STALL
```

Jsou-li splněny všechny požadavky, je možné provést žádost. Tedy odeslat číslo konfigurace, kterou zařízení aktuálně využívá. Tuto hodnotu umístíme do bufferu, nastavíme velikost odesílaných dat a ukazatel na data. V proměnné indikující současný stav zařízení nastavíme příznak započetí IN datové fáze a zavoláme rutinu pro odeslání dat. Tento postup je patrný z následujícího kódu.

```
lda     V_ConfigID
sta     Q_Tx_Buf                    ; aktualizuj USB Buffer Queue

mov     #$01,V_Tx_Size              ; velikost dat =1 byte
clr     V_Tx_Ptr

mov    #high(Q_Tx_Buf),V_Off_LDA_H
mov    #low(Q_Tx_Buf),V_Off_LDA_L   ; pointer na Q_Tx_Buf

bset    b_IN,V_Transact             ; IN transakce započata
jsr     TRAN_IN
```

5 Testovací úlohy

Princip testovacích úloh je v podstatě stejný. Nejprve je zapotřebí nalézt dané zařízení připojené k systému, získat k němu přístup a nechat si zaslat klíč prostřednictvím HID reportu. Další funkčnost již záleží na aplikaci. Jako testovací úlohu jsem implementoval jednoduchou konzolovou aplikaci v jazyce C++, která v přítomnosti hardwarového klíče vypíše na standardní výstup některé z informací uložených v USB klíči. Jako druhou testovací úlohu jsem se rozhodl implementovat jednoduchou kalkulačku s obráceným polským zápisem (RPN), provádějící jednoduché matematické operace. Kód tohoto kalkulátoru je převzat z volně dostupných příkladů v jazyce C#, převeden do jazyka C++ a obohacen o zamykání prostřednictvím hardwarového klíče. Pro úspěšné přeložení aplikací jsou zapotřebí dvě knihovny dostupné prostřednictvím Microsoft Development Kitu (HID.lib a SETUPAPI.lib). Obě aplikace pracují s jednoduchou třídou *USBKey*. Ta je vložena do jednotlivých projektů a zabezpečuje komunikaci s USB zařízením. Přeložené aplikace byly testovány pod třemi různými počítačovými stanicemi s operačním systém Windows XP. Sestavený hardwarový klíč byl rozpoznán jako HID zařízení v operačním systému Windows XP i linuxovém operačním systému Ubuntu.

V této kapitole bude popsaná nejdůležitější metoda třídy *USBKey* `FindHID()`, která slouží pro nalezení daného USB zařízení. Implementace ostatních metod jsou relativně jednoduché a proto není potřeba, aby zde byly uvedeny.

5.1 Nalezení USB zařízení

Nalezení navrženého USB zařízení je implementováno prostřednictvím metody `FindHID()`, jak bylo uvedeno výše. Prvním krokem je získání jednoznačného ukazatele na HID zařízení. Toho je dosaženo prostřednictvím API funkce `HidD_GetHidGuid()`, která jako parametr vyžaduje referenci na strukturu GUID. GUID je v podstatě 128bitový integer, který slouží pro jednoznačnou identifikaci.

```
HidD_GetHidGuid(&HidGuid);
```

Tento získaný ukazatel využijeme ve funkci `SetupDiGetClassDevs()`, která vrací handler na informace o všech zařízeních nainstalovaných v systému. Pomocí příznaků DIGCF_PRESENT a DIGCF_INTERFACEDEVICE sdělíme API funkci požadavek na zařízení, která jsou aktuálně přítomna v operačním systému a podporují rozhraní (Device interfaces). Jestliže se nám podaří získat tento handler, stačí postupně procházet daná zařízení, dokud nenalezneme to pravé.

```
hDevInfo=SetupDiGetClassDevs(&HidGuid, NULL, NULL,
                                DIGCF_PRESENT|DIGCF_INTERFACEDEVICE)
devInfoData.cbSize = sizeof(devInfoData);
```

Abychom nalezli všechna rozhraní, která jsou přítomna v informační sadě (device information set) *hDevInfo*, využijeme API funkce `SetupDiEnumDeviceInterfaces()`. Po návratu z této funkce obsahuje proměnná *devInfoData* strukturu `SP_DEVICE_INTERFACE_DATA`. V této struktuře jsou obsažena všechna rozhraní, odpovídající zadaným parametrům. Funkce v parametrech vyžaduje handler vrácený funkcí `SetupDiGetClassDevs()`, GUID vrácený funkcí `HidD_GetHidGuid()` a index specifikující zařízení. Pokud vše proběhlo bez chyb, je vrácena logická hodnota TRUE.

```
Result=SetupDiEnumDeviceInterfaces(hDevInfo, 0, &HidGuid, Device,
                    &devInfoData);
```

Abychom zjistili podrobné informace týkající se rozhraní hledaného zařízení, musíme dvakrát zavolat funkci `SetupDiGetDeviceInterfaceDetail()`. Poprvé pro zjištění velikosti této struktury předanou parametru *Length* a po druhé pro obdržení detailních informací o rozhraní zařízení ve struktuře `PSP_DEVICE_INTERFACE_DETAIL_DATA` předanou jako parametr této funkci v proměnné *detailData*. Funkce vyžaduje DeviceInfoSet *hDevInfo* vrácený funkcí `SetupDiGetClassDevs()` a strukturu `SP_DEVICE_INTERFACE_DATA` specifikující dané rozhraní vrácenou funkcí `SetupDiEnumDeviceInterfaces()`.

```
Result = SetupDiGetDeviceInterfaceDetail(hDevInfo, &devInfoData,
                          NULL, 0, &Length, NULL);
detailData = (PSP_DEVICE_INTERFACE_DETAIL_DATA)malloc(Length);
detailData -> cbSize = sizeof(SP_DEVICE_INTERFACE_DETAIL_DATA);

Result = SetupDiGetDeviceInterfaceDetail(hDevInfo, &devInfoData,
                           detailData, Length, &Required, NULL);
```

Nyní potřebujeme zařízení otevřít pro čtení. K tomu slouží funkce `CreateFile()`, umožňující čtení a zápis do souborů, adresářů, fyzických disků, komunikačních zdrojů a rour. Funkce vyžaduje cestu ke zdroji (v našem případě obsaženou v proměnné *detailData->DevicePath*), parametry přístupu k objektu (GENERIC_READ), režim sdílení objektů (FILE_SHARE_READ), bezpečnostní atributy, co provést v případě existence nebo neexistence souboru (OPEN_EXISTING) a příznaky.

```
DeviceHandle=CreateFile(detailData->DevicePath, GENERIC_READ,
                    FILE_SHARE_READ, NULL,OPEN_EXISTING, 0, NULL);
```

Pro jednoznačnou identifikaci zařízení slouží VendorID a ProductID. Pro získání těchto dvou identifikačních prvků využijeme funkce `HidD_GetAttributes()`, která vyžaduje handler vrácený funkcí `CreateFile()`.

```
Attributes.Size = sizeof(Attributes);
Result = HidD_GetAttributes(DeviceHandle, &Attributes);
```

Nyní mužeme jednoznačně určit zda se jedná o naše zařízení. Jestliže nikoliv, je dobré získané handlery uzavřít prostřednictvím funkce `CloseHandle()`.

```
if (Attributes.VendorID == VendorID)
{
      if (Attributes.ProductID == ProductID)
      {
            DeviceDetected = TRUE;
      }
      else
            CloseHandle(DeviceHandle);
}
else
      CloseHandle(DeviceHandle);
```

5.2 Zaslání klíče

Před zasláním klíče je nutné nejprve zjistit, zda se v systému nachází dané zařízení prostřednictvím metody `FindHID()`. Pro zaslání reportu využijeme funkce `HidD_GetFeature()`, která jako parametry vyžaduje handler vrácený funkcí `CreateFile()`, buffer, do kterého se má zapisovat a jeho velikost. Do toho bufferu funkce uloží příchozí report (v našem případě klíč uložený v USB zařízení). Po té mužeme jednoduše přijatý klíč přečíst a rozhodnout o dalším chodu aplikace.

6 Závěr

Tato bakalářská práce se zabývá návrhem a tvorbou USB zařízení s využitím rozhraní nabízeného mikrokontrolérem firmy Motorola MC68HC908JB8. Práce se snaží ukázat, jak je možné propojit navrhované zařízení spolu s operačním systémem Windows a jak je relativně snadné s ním komunikovat prostřednictvím jednoduchých funkcí nabízených jádrem operačního systému pro komunikaci s HID zařízeními.

Firmware byl vyvinut především pro mikrokontrolér MC68HC908JB8, ale přenositelnost je možná i na jiné typy mikrokontrolérů rodiny HC08 podporující stejný USB modul. Prakticky to však nebylo ověřeno.

Díky možnosti programování na čipu, kterou mikrokontroléry HC08 nabízejí, bylo zařízení rychle sestrojeno a snadno naprogramováno. S využitím vývojového kitu JANUS bylo možné vyvíjet firmware na libovolném počítači vybaveným sériovým portem. Tyto fakty dokazují, proč jsou mikrokontroléry HC08 a jejich nástupci s jádrem HCS08 hojně využívány po celém světě.

Práce mě obohatila o znalosti týkající se standardu USB a hardwarových zařízení. Z počátku přinesla úskalí spojená s objemnou USB specifikací a specifikací HID. Výrazně mi ulehčily práci různé výtahy z těchto specifikací vytvořené pro rychlý a efektivní návrh. Po prvotních těžkých krocích a pochopení principu USB komunikace implementované v modulu mikrokontroléru jsem však již nenarazil na vážnější komplikace. Z počátku obtížné dodržování rozsáhlých standardů HID se vyplatilo při vývoji software na cílové platformě. Nebylo nutné vytvářet vlastní ovladač, ale i přesto jsem získal cenné informace o odstínění moderních operačních systému od samotného hardwaru.

Hardwarový klíč jako takový je jednoduché zařízení a tudíž většina možných implementačních vylepšení se týká především softwaru, který s ním pracuje. I přesto je možné provést některá drobná rozšíření. U hardwarového návrhu dosáhneme miniaturizace odstraněním vnějšího pull-up rezistoru, který je již obsažen na čipu mikrokontroléru, součástek u vývodu IRQ a diod, které byly využity pro testovací účely. Firmware uložený v zařízení plně nevyužívá kapacity paměti a koncových bodů, které daná platforma nabízí. Je tedy možné nahradit použitý mikrokontrolér levnějším s menší pamětí a menší energetickou spotřebou. Funkčnost byla ověřena na jednoduchých testovacích úlohách běžících pod operačním systémem Windows. Díky univerzálnosti HID specifikace je možné vytvořit složitější aplikace pod různými operačními systémy (Linux, FreeBSD, Mac OS X) například uzamčení celého PC namísto jediné aplikace.

Literatura

[1] Hyde, J.: USB design by example. 2, USA, Intel Press 2001.

[2] Freescale Semiconductor: MC68HC908JB8 Technical Data. 2005. Dokument dostupný na URL http://www.freescale.com/files/microcontrollers/doc/data_sheet/MC68HC908JB8.pdf (květen 2008).

[3] Freescale Semiconductor: USB08 Universal Serial Bus Evolution Board Using the MC68HC908JB8. 2001. Dokument dostupný na URL http://hc08web.de/usb08/files/drm002.pdf (květen 2008).

[4] Beyondlogic: USB in a Nutshell. 2002. Dokument dostupný na URL http://www.beyondlogic.org/usbnutshell/usb1.htm (květen 2008).

[5] USB.org: Device Class Definition for Human Interface Device (HID). 2001. Dokument dostupný na URL http://www.usb.org/developers/devclass_docs/HID1_11.pdf (květen 2008).

[6] Freescale Semiconductor: CPU08 Central Procesor Unit Reference Manual. 2006. Dokument dostupný na URL http://www.freescale.com/files/microcontrollers/doc/ref_manual/CPU08RM.pdf (květen 2008).

[7] USB.org: Official site. Includes information, products, developers section, and press. 2008. Dokument dostupný na URL http://www.usb.org/home (květen 2008).

[8] Schwarz, J.; Růžička, R.; Strnadel, J.: Studijní opora k předmětu Mikroprocesorové a vestavěné systémy. Brno, FIT VUT 2006.

[9] Náprstek, J.: Vývojový kit JANUS – konstrukční manuál. Praha, Motorola s.r.o. 2003.

[10] Freescale Semiconductor: Information on Digital Signal Processors from Freescale Semiconductor (Motorola). 2008. Dokument dostupný na URL http://www.freescale.com/ (květen 2008).

[11] HW.cz: Vše o elektronice a programování. 2008. Dokument dostupný na URL http://hw.cz/ (květen 2008).

[12] MSDN: Offers help for developers in writing applications using Microsoft products. 2008. Dokument dostupný na URL msdn2.microsoft.com/ (květen 2008).

[13] Prosise, J.: Microsoft .NET Webové aplikace v .NET Framework, C# a ASP.NET. Brno, Computer Press 2003.

[14] Axelson, J.: The HID Page, Resources for Developers of USB Devices in the Human Interface Device Class. Dokument dostupný na URL http://www.lvr.com/hidpage.htm (květen 2008).

[15] Corbet, J.; Rubini, A.; Kroah-Hartman, G.: Linux Device Drivers, Third Edition, 2005. Dokument dostupný na URL http://lwn.net/Kernel/LDD3/ (květen 2008).

[16] Microsoft: Main site for product information, support, and news. 2008. Dokument dostupný na URL http://www.microsoft.com/Downloads/ (květen 2008).

Seznam příloh

Příloha A

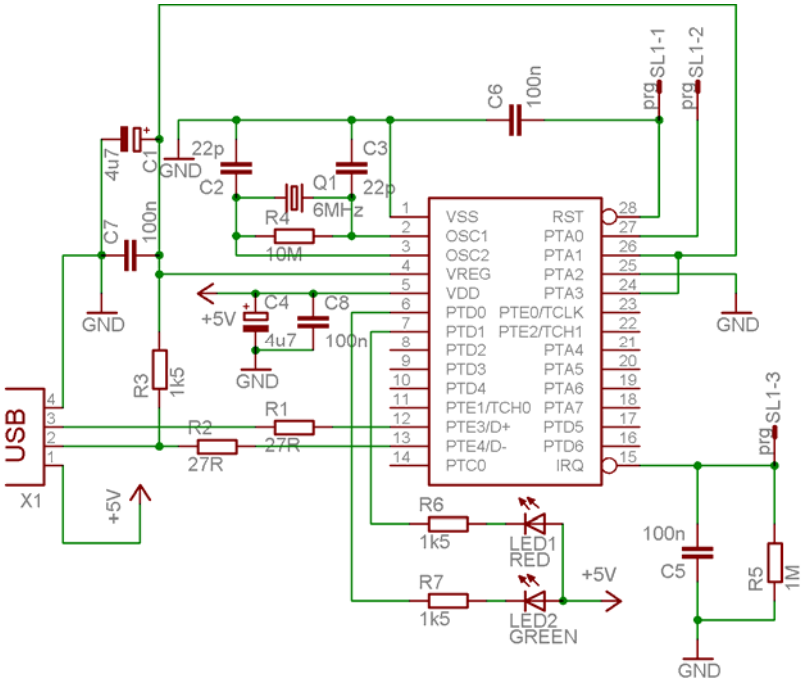
Obsah CD

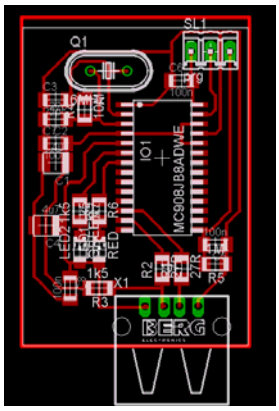
Na přiloženém CD se nachází následující soubory a adresáře:

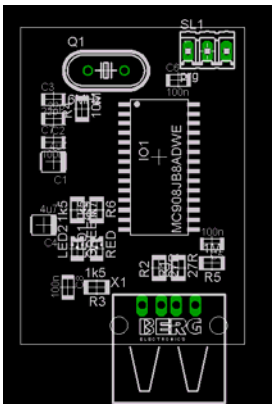
- Soubor USBKey.pdf – technická zpráva ve formátu PDF.
- Adresář USBKey – projekt vytvořený v Codewarrior V5.1 obsahující zdrojové texty firmwaru USB klíče.
- Adresář LED – projekt vytvořený v Codewarrior V5.1 obsahující zdrojové texty pro jednoduchý blikač sloužící k otestovaní funkčnosti mikrokontroléru.
- Adresář Software – obsahující zdrojové texty pro konzolovou aplikaci v jazyce C++ vypisující informace o USB klíči do standardního okna a grafickou aplikaci v podobě RPN kalkulátoru s polskou notací uzamykatelnou prostřednictvím USB klíče obě vytvořené jako projekty ve Visual Studiu 2005.
- Adresář Doc – obsahující volně šiřitelé dokumenty týkající se USB specifikace, HID specifikace a datasheetů společnosti Freescale.
- Adresář Schéma – obsahující schéma a návrh plošných spojů vytvořených v programu Eagle 4.16.

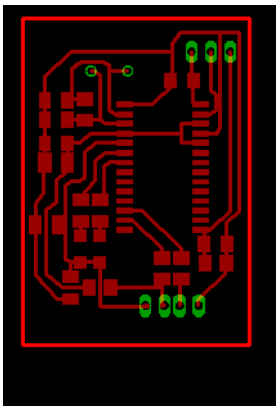
Příloha B

Schémata zapojení



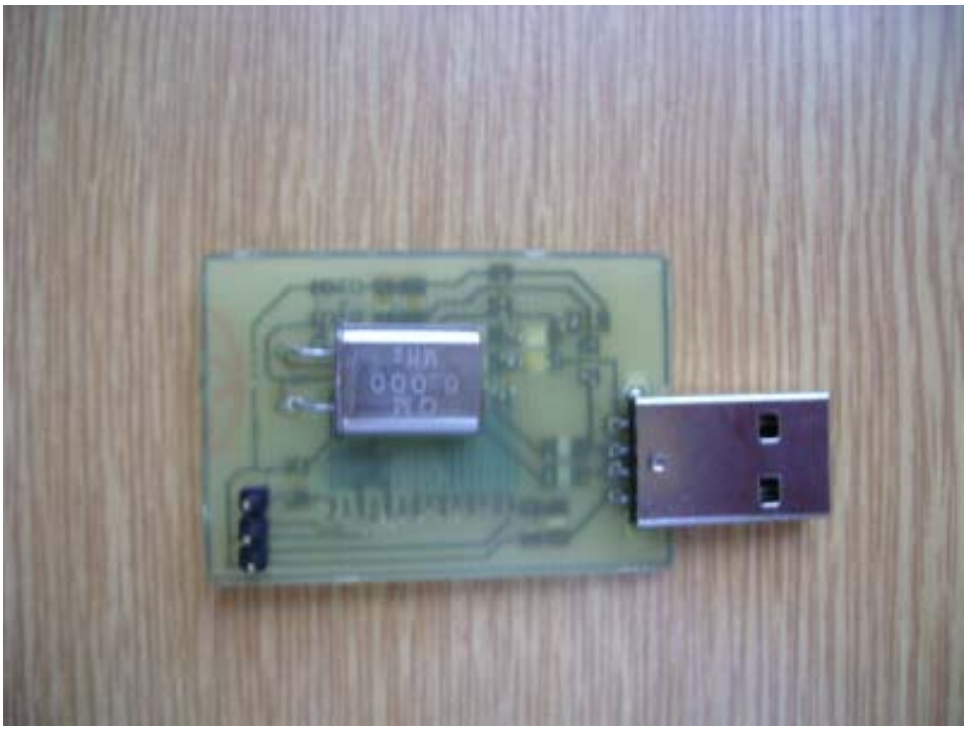


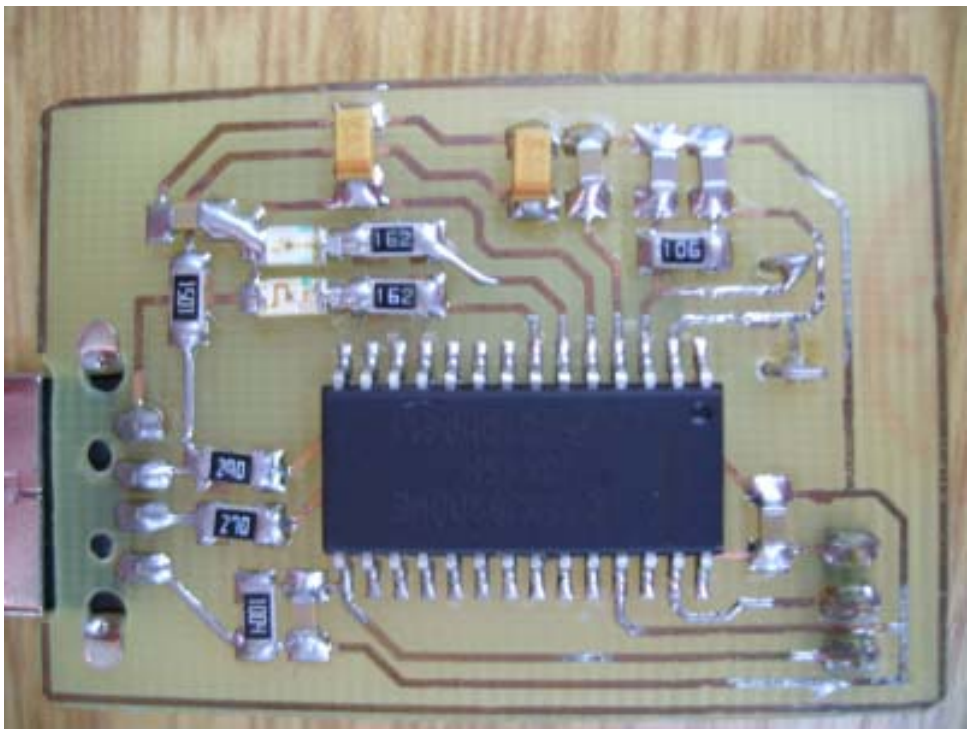




Příloha C

Konstrukce zařízení





Příloha D

Ukázky aplikací